# 「付録1」
# セルフコンパイラでkernel再構築

コンパイラせずに再構築をするなら rpi-updateコマンドを使う。

まず、再構築前のkernelのバージョンを確認する。

$uname　-a

```
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 4.19.57-v7+ #1244 SMP
pi@raspberrypi:~ $
```

$ sudo rpi-update （ただし、プロキシ環境ではフリーズする）

```
pi@raspberrypi:~ $ sudo rpi-update
```

プロキシ環境では失敗するので　^C　で中断。

また、rpi-updateによるkernelの再構築の場合、「linux」フォルダのダウンロードが不要なので「linux」フォルダは生成されない。

# 「付録1」
# セルフコンパイラでkernel再構築

以下は rpi-update をプロキシ環境で使う場合なので対象の場合のみ参照。
ネット情報に従い、多少処理に手間がかかる。
/usr/bin/にrpi-updateが確認できるが、ユーザ「pi」には書き込み権限がない。

```
pi@raspberrypi:~ $ cd /usr/bin
pi@raspberrypi:/usr/bin $ ls -l rpi-update
-rwxr-xr-x 1 root root 7037  7月  8  2014 rpi-update
pi@raspberrypi:/usr/bin $
```

オリジナルのrpi-updateには触れず、rpi-updateを「pi」ホームへコピーする。

```
pi@raspberrypi:/usr/bin $ pwd
/usr/bin
pi@raspberrypi:/usr/bin $ cd
pi@raspberrypi:~ $ pwd
/home/pi
pi@raspberrypi:~ $ cp /usr/bin/rpi-update ./
pi@raspberrypi:~ $ ls -l rpi-update
-rwxr-xr-x 1 pi pi 7037  8月  1 10:40 rpi-update
pi@raspberrypi:~ $
```

# 「付録1」
# セルフコンパイラでkernel再構築

rpi-updateの内容を修正して、rpi-update-proxy　とする。

ここでは、プロキシサーバIPを172.16.0.8、ポートを8080とする。

sedコマンドを使って rpi-updateファイル内の文字列を以下のように置き換え、ファイル名をrpi-update-proxyにする。

$sed　-e　“s/curl/curl　-x　172.16.0.8:8080/”　rpi-update > rpi-update-proxy

```
pi@raspberrypi:~ $ sed -e "s/curl/curl -x 172.16.0.8:8080/"
 rpi-update > rpi-update-proxy
```

作成後のrpi-update-proxyのパーミションには実行権限がない。

```
pi@raspberrypi:~ $ ls -l rpi-update-proxy
-rw-r--r-- 1 pi pi 7113  8月  1 10:45 rpi-update-proxy
pi@raspberrypi:~ $
```

# 「付録1」
# セルフコンパイラでkernel再構築

実行権限を付加する。

```
pi@raspberrypi:~ $ chmod a+x rpi-update-proxy
pi@raspberrypi:~ $ ls -l rpi-update-proxy
-rwxr-xr-x 1 pi pi 7113  8月  1 10:45 rpi-update-proxy
pi@raspberrypi:~ $
```

再構築時、rpi-update自身が最新版に置き換えさせないようにするには、
「UPDATE_SELF=0」のオプションを付けて実行する

```
pi@raspberrypi:~ $ sudo UPDATE_SELF=0 ./rpi-update-proxy
 *** Raspberry Pi firmware updater by Hexxeh, enhanced by A
 *** We're running for the first time
 *** Backing up files (this will take a few minutes)
 *** Backing up firmware
 *** Backing up modules 4.19.57-v7+
```

再起動後、バージョンを確認する。

```
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 4.19.60-v7+ #1247 SMP
pi@raspberrypi:~ $
```

# 「付録1」
# セルフコンパイラでkernel再構築

次にmoduleを使ってセルフコンパイラでのkernelの再構築を紹介する。モジュールまたはカーネルヘッダはドライバの作成には必要になる。

まずターミナルを起動し、gitにて「linux」をダウンロードする。（環境が代わって以降、gitは使えなくなった）

CUI環境のプロキシ越え処理を施し（②で処理済）

$ git clone –depth=1 https://github.com/raspberrypi/linux

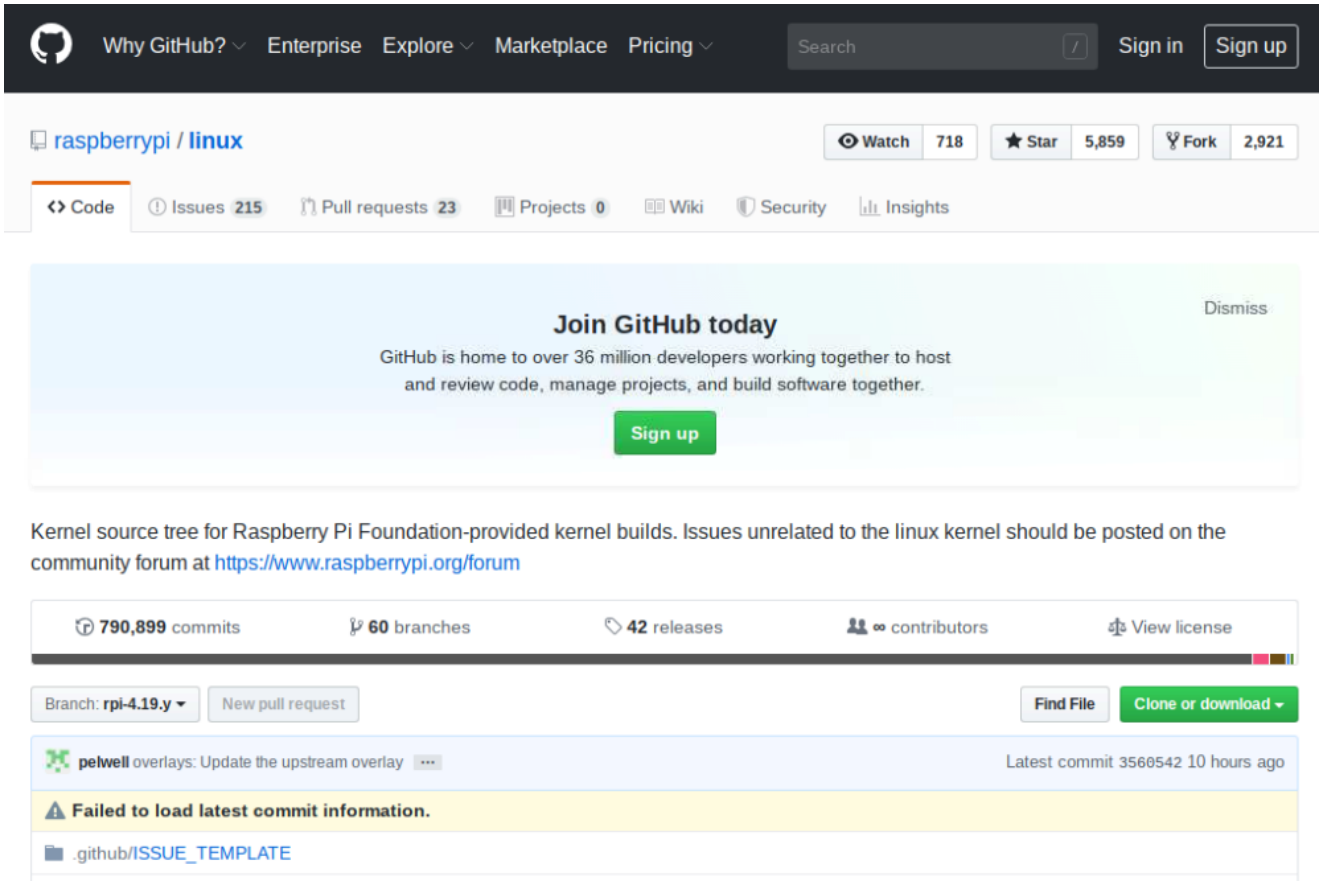
```
pi@raspberrypi:~ $ git clone --depth=1 https://github.com/raspberrypi/linux
Cloning into 'linux'...
remote: Enumerating objects: 65788, done.
remote: Counting objects: 100% (65788/65788), done.
remote: Compressing objects: 100% (60315/60315), done.
Receiving objects:  78% (51699/65788), 140.19 MiB | 484.00 KiB/s
```

```
pi@raspberrypi:~ $ ls ./linux/
COPYING         Kconfig      arch    drivers   init    mm        security  virt
CREDITS         MAINTAINERS  block   firmware  ipc     net       sound
Documentation   Makefile     certs   fs        kernel  samples   tools
Kbuild          README       crypto  include   lib     scripts   usr
pi@raspberrypi:~ $
```

# 「付録1」
# セルフコンパイラでkernel再構築

もしくはwebブラウザを使ってダウンロードする。

https://github.com/raspberrypi/linux



# 「付録1」
# セルフコンパイラでkernel再構築

ダウンロードしたZIPファイルを展開したフォルダを”linux”として適当な場所に保存する。

ここでは ~/linux フォルダとして使用する。

```
pi@raspberrypi:~ $ mv ./ダウンロード/linux-rpi-4.19.y.zip ./
pi@raspberrypi:~ $ unzip linux-rpi-4.19.y.zip
```

解凍(15分ほど)したフォルダ名はlinux-rpi-4.14.y でこの名前を変更する。

$ mv  linux-rpi-4.14.y  linux

```
pi@raspberrypi:~ $ ls linux*
linux-rpi-4.19.y.zip

linux-rpi-4.19.y:
COPYING         Kconfig      README  crypto    include  lib      scripts   usr
CREDITS         LICENSES     arch    drivers   init     mm       security  virt
Documentation   MAINTAINERS  block   firmware  ipc      net      sound
Kbuild          Makefile     certs   fs        kernel   samples  tools
pi@raspberrypi:~ $ mv linux-rpi-4.19.y linux
```

# 「付録1」
# セルフコンパイラでkernel再構築

bc, libncurses5-dev,をインストールする。linux14.19頃からはbison, flex, libssl-dev を追加インストールする。

$ sudo apt-get install -y bc libncurses5-dev bison flex libssl-dev

```
pi@raspberrypi:~ $ sudo apt-get install -y bc libncurses5-dev
 bison flex libssl-dev
```

# 「付録1」
# セルフコンパイラでkernel再構築

ダウンロードしたlinuxフォルダに移動する。

KERNELはRaspi2または3は”kernel7”であり、”bcm2709_defconfig”を指定する。

bcm2835_defconfig, bcmrpi_defconfig　はそのままでは適用できない。

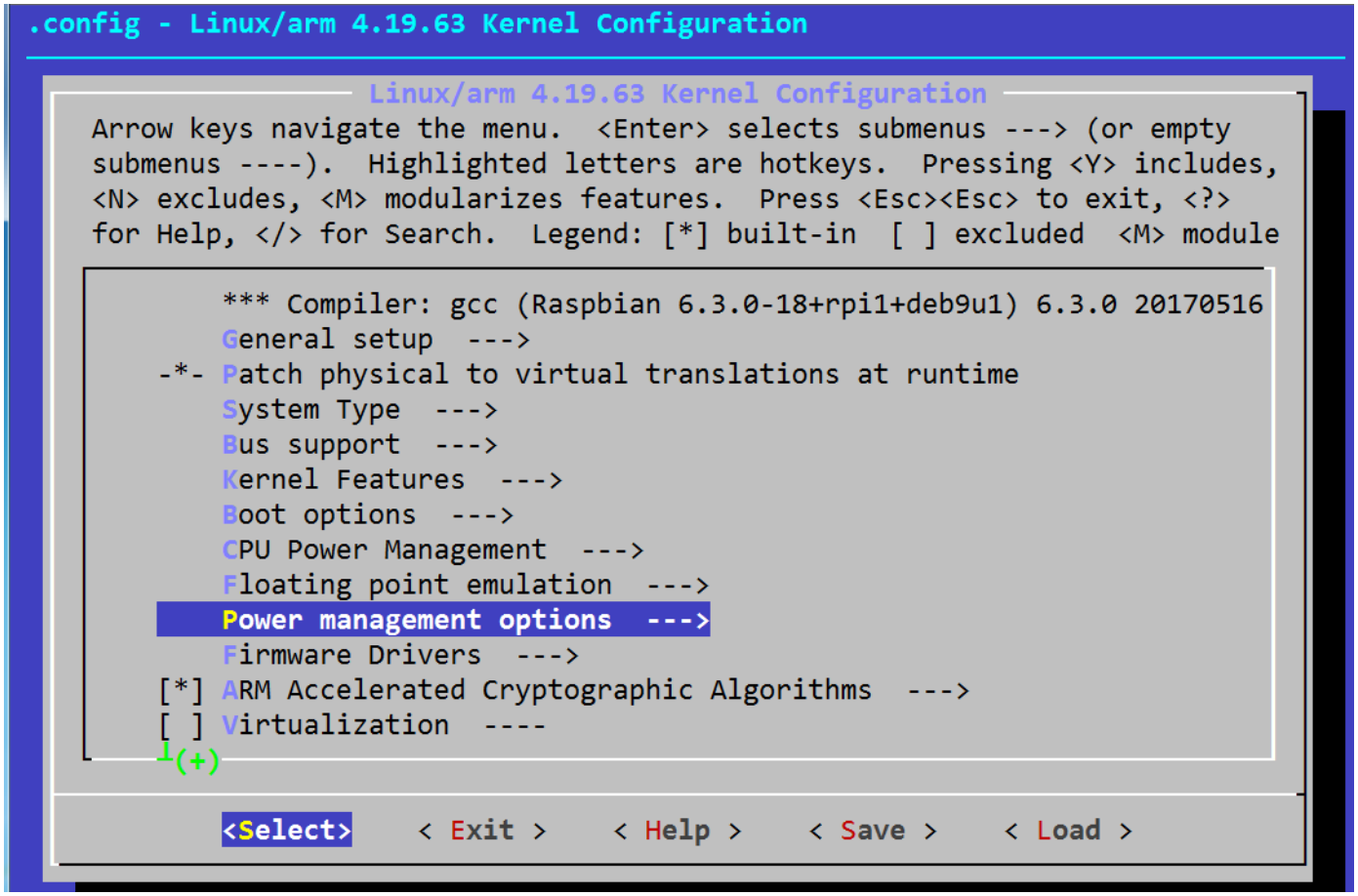
$ make bcm2709_defconfig　を実行して.config を生成する。

```
pi@raspberrypi:~ $ cd linux
pi@raspberrypi:~/linux $ KERNEL=kernel7
pi@raspberrypi:~/linux $ make bcm2709_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  YACC    scripts/kconfig/zconf.tab.c
  LEX     scripts/kconfig/zconf.lex.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
pi@raspberrypi:~/linux $
```

# 「付録1」
# セルフコンパイラでkernel再構築

menuconfigで確認する。

```
pi@raspberrypi:~/linux $ make menuconfig
```



# 「付録１」
# セルフコンパイラでkernel再構築

以降のbuildとinstallには相当な時間を要する。(makeに５～６時間要する)

-j4 オプションが使えれば　Model2---およそ３時間で終了

Model3---およそ２時間で終了

$ make （-j4） zImage modules dtbs

```
pi@raspberrypi:~/linux $ make -j4 zImage modules dtbs
```

# 「付録1」
# セルフコンパイラでkernel再構築

セルフコンパイルでも最も時間がかかると思われるコンパイル量でも2時間程度で終了できるようになった。

クロスコンパイラはほぼ必要の無いレベルとも考えられる。

linuxフォルダ内には「vmlinux」が生成される。

```
  LD [M]  sound/usb/misc/snd-ua101.ko
  LD [M]  sound/usb/snd-usb-audio.ko
  LD [M]  sound/usb/snd-usbmidi-lib.ko
pi@raspberrypi:~/linux $ ls
COPYING         Makefile         certs     ipc              samples   vmlinux
CREDITS         Module.symvers   crypto    kernel           scripts   vmlinux.o
Documentation   README           drivers   lib              security
Kbuild          System.map       firmware  mm               sound
Kconfig         arch             fs        modules.builtin  tools
LICENSES        block            include   modules.order    usr
MAINTAINERS     built-in.a       init      net              virt
pi@raspberrypi:~/linux $ ls -l
```

# 「付録1」
# セルフコンパイラでkernel再構築

/arch/arm/boot/フォルダ内には「dts」フォルダや「zImage」が生成されている。

```
pi@raspberrypi:~/linux $ cd arch/arm/boot
pi@raspberrypi:~/linux/arch/arm/boot $ ls
Image      bootp       deflate_xip_data.sh  install.sh
Makefile   compressed  dts                  zImage
pi@raspberrypi:~/linux/arch/arm/boot $
```

# 「付録1」
# セルフコンパイラでkernel再構築

インストール前のkernelのバージョンアップをunameで確認する。

$ uname　−a

```
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 4.19.57-v7+ #1244
pi@raspberrypi:~ $
```

ここでは、4.19.57−v7+ となっている。

念のため　KERNEL登録を確認する

$ KERNEL=kernel7

```
pi@raspberrypi:~ $ KERNEL=kernel7
```

# 「付録1」
# セルフコンパイラでkernel再構築

モジュールをインストールする。

$ sudo make modules_install

```
pi@raspberrypi:~ $ cd linux/
pi@raspberrypi:~/linux $ sudo make modules_install
```

```
  INSTALL sound/usb/caiaq/snd-usb-caiaq.ko
  INSTALL sound/usb/hiface/snd-usb-hiface.ko
  INSTALL sound/usb/misc/snd-ua101.ko
  INSTALL sound/usb/snd-usb-audio.ko
  INSTALL sound/usb/snd-usbmidi-lib.ko
  DEPMOD  4.19.63-v7
pi@raspberrypi:~/linux $
```

# 「付録1」
# セルフコンパイラでkernel再構築

生成されたファイル類をコピーする。

$ sudo cp arch/arm/boot/dts/*.dtb  /boot/

$ sudo cp arch/arm/boot/dts/overlays/*.dtb*  /boot/overlays/

$ sudo cp arch/arm/boot/dts/overlays/README  /boot/overlays/

```
pi@raspberrypi:~/linux $ sudo cp arch/arm/boot/dts/*.dtb /boot
pi@raspberrypi:~/linux $ sudo cp arch/arm/boot/dts/overlays/*.dtb* /boot/overlays/
pi@raspberrypi:~/linux $ sudo cp arch/arm/boot/dts/overlays/README /boot/overlays/
pi@raspberrypi:~/linux $
```

現行のイメージファイルのバックアップが必要な場合は、保存しておく。

ここでは、$KERNELは“kernel7”の名前の置き換えなので

kernel7-backup.imgファイル名で保存する。

```
pi@raspberrypi:~/linux $ sudo cp /boot/$KERNEL.img /boot/$KERNEL-backup.img
pi@raspberrypi:~/linux $
```

# 「付録1」
# セルフコンパイラでkernel再構築

mkknlimgスクリプトコマンドで生成されたイメージファイルと入替える。

$ sudo scripts/mkknlimg  arch/arm/boot/zImage  /boot/$KERNEL.img

```
pi@raspberrypi:~/linux $ sudo scripts/mkknlimg arch/arm/boot/zImage
 /boot/$KERNEL.img
Version: Linux version 4.19.63-v7 (pi@raspberrypi) (gcc version 6.3.0
 20170516 (Raspbian 6.3.0-18+rpi1+deb9u1)) #1 SMP Thu Aug 1 16:44:18
JST 2019
DT: y
DDT: y
270x: y
283x: y
pi@raspberrypi:~/linux $
```

# 「付録1」
# セルフコンパイラでkernel再構築

Raspiを再起動する。

カーネルのバージョンを確認する。

```
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 4.19.63-v7 #1 SMP
pi@raspberrypi:~ $
```

バージョンが 4.19.63-v7 になる。

# 「付録2」
# クロスコンパイラでkernel再構築

別の方法としてubuntuPCのクロスコンパイル環境でカーネルのインストールをする。

安全を見てNFSで処理せず、RaspberryPiターゲットからSDカードを抜いて処理する。SDカードリーダライタでPC側のLinuxOSのubuntuに認識させる。ここでは/media/user01にSDカードがマウントされることになる。

まず、ここでの現状のバージョンはRaspberryPiターゲットにて

$uname　−a

```
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 4.19.57-v7+ #1244 SMP
pi@raspberrypi:~ $
```

# 「付録2」
# クロスコンパイラでkernel再構築

開発支援PCでは /media/user01にSDカードがマウントされることになる。

まずは　/media/user01 フォルダに 「SETTING」と「boot」と「root」の各フォルダが確認できる。

```
user01@ubuntu:~$ cd /media/user01/
user01@ubuntu:/media/user01$ ls
SETTINGS  boot  root
user01@ubuntu:/media/user01$
```

続けて　df　−Tコマンドでも確認すると。

$ df −T

```
user01@ubuntu:/media/user01$ df -T
```

# 「付録2」
# クロスコンパイラでkernel再構築

以下からわかるように/bootフォルダは/dev/sdd6パーティションがマウントされており、ファイルシステムは「vfat」すなわち「fat32」、/SETTINGSフォルダは/dev/sdd5パーティションがマウントされており、同、「ext4」タイプで、/rootフォルダは/dev/sdd7パーティションがマウントされており、同、「ext4」タイプである。

```
user01@ubuntu:/media/user01$ df -T
Filesystem     Type     1K-blocks     Used Available Use% Mounted on
udev           devtmpfs    979260        0    979260   0% /dev
tmpfs          tmpfs       201736     1776    199960   1% /run
/dev/sda1      ext4      51341792 29204144  19499928  60% /
tmpfs          tmpfs      1008664        0   1008664   0% /dev/shm
tmpfs          tmpfs         5120        4      5116   1% /run/lock
tmpfs          tmpfs      1008664        0   1008664   0% /sys/fs/cgroup
tmpfs          tmpfs       201732       16    201716   1% /run/user/120
tmpfs          tmpfs       201732       28    201704   1% /run/user/1000
/dev/sdd5      ext4         30701      398     28010   2% /media/user01/SETTINGS
/dev/sdd6      vfat         69553    22192     47362  32% /media/user01/boot
/dev/sdd7      ext4      13032168  4415944   7931168  36% /media/user01/root
/dev/loop0     squashfs     88704    88704         0 100% /snap/core/4486
user01@ubuntu:/media/user01$
```

# 「付録2」
# クロスコンパイラでkernel再構築

クロス環境で本文にあったkernelコンパイルを済ませたlinuxフォルダに移動する。(ここでは本文処理済の　~/raspi/linux)

念のため　KERNEL=kernel7(Model2またはModell3の場合)を再度設定し、

```
user01@ubuntu:~/raspi/linux$ KERNEL=kernel7
```

makeコマンドにてモジュールをext4(ここではrootフォルダに相当)にインストールする。また、make時、CROSS_COMPILE=. . . はフルパスにする必要がある。

(ここではクロス環境の/toolsがルートにあるとしている。)

```
user01@ubuntu:~/raspi/linux$ sudo make ARCH=arm CROSS_COMPILE=/tools/
arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin/arm-linux
-gnueabihf- INSTALL_MOD_PATH=/media/user01/root modules_install
```

# 「付録2」
# クロスコンパイラでkernel再構築

ここでは　4.19.60-v7 がインストールされた。

```
  INSTALL sound/usb/misc/snd-ua101.ko
  INSTALL sound/usb/snd-usb-audio.ko
  INSTALL sound/usb/snd-usbmidi-lib.ko
  DEPMOD  4.19.60-v7
user01@ubuntu:~/raspi/linux$
```

現行のカーネルイメージを保存・バックアップしたい場合、別名でコピーする。

ここではkernel7-backup.imgとしてコピーする。

```
user01@ubuntu:~/raspi/linux$ sudo cp /media/user01/boot/$KERNEL.img
/media/user01/boot/$KERNEL-backup.img
```

# 「付録2」
# クロスコンパイラでkernel再構築

必要なデータをmkknlimg コマンドや cp コマンドでコピーする。

```
user01@ubuntu:~/raspi/linux$ sudo scripts/mkknlimg arch/arm/boot/zImage
 /media/user01/boot/$KERNEL.img
Version: Linux version 4.19.60-v7 (user01@ubuntu) (gcc version 4.8.3 201
40303 (prerelease) (crosstool-NG linaro-1.13.1+bzr2650 - Linaro GCC 2014
.03)) #1 SMP Thu Aug 1 10:10:52 JST 2019
DT: y
DDT: y
270x: y
283x: y
user01@ubuntu:~/raspi/linux$ sudo cp arch/arm/boot/dts/*dtb
 /media/user01/boot/
user01@ubuntu:~/raspi/linux$ sudo cp arch/arm/boot/dts/overlays/*.dtb*
 /media/user01/boot/overlays/
user01@ubuntu:~/raspi/linux$ sudo cp arch/arm/boot/dts/overlays/README
 /media/user01/boot/overlays/
user01@ubuntu:~/raspi/linux$
```

# 「付録2」
# クロスコンパイラでkernel再構築

SDカードをRaspberryPiターゲットにとりつける。

$ uname –a にてバージョンが確認できる。

```
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 4.19.60-v7 #1 SMP
pi@raspberrypi:~ $
```

ここでは　4.19.60–v7になっていることが確認できる。

# 「付録３」

セルフだと2時間かかるコンパイルをクロスコンパイルで試す。

必要なインストールデータをPC側にダウンロードする。

必要なデータは「ffmpeg」と「x264」、「alsa-lib」でダウンロードする。

まず、PC側の公開フォルダである　/publicフォルダに/workffmpegフォルダを作りその配下に必要データをダウンロードする。

```
user01@ubuntu:~$ cd /public
user01@ubuntu:/public$ mkdir workffmpeg
user01@ubuntu:/public$ cd workffmpeg
user01@ubuntu:/public/workffmpeg$
```

新しいプロキシ環境ではgitが使えないので③を参照して「ffmpeg」、「x264」、「alsa-lib」を入手する。

```
user01@ubuntu:~/ダウンロード$ ls
FFmpeg-master.zip      tools-master.zip
linux-rpi-4.19.y.zip   x264-master.zip
user01@ubuntu:~/ダウンロード$
```

# 「付録３」

セルフだと2時間かかるコンパイルをクロスコンパイルで試す。

それぞれを解凍して所定のフォルダに所定の名前で移動させる。

FFmpegについて

```
user01@ubuntu:~/ダウンロード$ unzip FFmpeg-master.zip
```

```
user01@ubuntu:~/ダウンロード$ mv FFmpeg-master /public/workffmpeg/ffmpeg
```

X264について

```
user01@ubuntu:~/ダウンロード$ unzip x264-master.zip
```

```
user01@ubuntu:~/ダウンロード$ mv x264-master /public/workffmpeg/x264
```

# 「付録3」

続いて、サウンド機能の　alsa-libはwgetにてダウンロードし、解凍する

```
user01@ubuntu:/public/workffmpeg$ wget
 ftp://ftp.alsa-project.org/pub/lib/alsa-lib-1.1.9.tar.bz2
```

```
user01@ubuntu:/public/workffmpeg$ ls
alsa-lib-1.1.9.tar.bz2  ffmpeg  x264
user01@ubuntu:/public/workffmpeg$
```

```
user01@ubuntu:/public/workffmpeg$ tar xvjf alsa-lib-1.1.9.tar.bz2
```

フォルダ内には3つの関連フォルダ

```
alsa-lib-1.1.9/ltmain.sh
alsa-lib-1.1.9/aclocal.m4
alsa-lib-1.1.9/configure.ac
alsa-lib-1.1.9/NOTES
user01@ubuntu:/public/workffmpeg$ ls
alsa-lib-1.1.9  alsa-lib-1.1.9.tar.bz2  ffmpeg  x264
user01@ubuntu:/public/workffmpeg$
```

# 「付録3」

ubuntuPCからのインストール先は　NFSを通したターゲット。つまりRaspiの /usr/local/　である。

したがってRaspi側の公開フォルダは/usr/local　として以下のように追加する。

rootインストールできるように　no_root_squash を入れる。クライアント指定行では、「スペース」は併記とみなされるので「スペース」を入れない。

```
pi@raspberrypi:~ $ sudo nano /etc/exports
```

```
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#

/srv/nfs 192.168.137.0/24(rw,sync,no_subtree_check)
/usr/local 192.168.137.0/24(rw,sync,no_subtree_check,no_root_squash)
```

内容を反映させ、Raspiのサーバを起動する。サービスの再起動は

$ sudo　systemctl　restart　nfs-kernel-server(.service)

```
pi@raspberrypi:~ $ sudo exportfs -a
pi@raspberrypi:~ $ sudo systemctl restart nfs-kernel-server.service
```

# 「付録3」

PCにてRaspi(ここでは192.168.137.55)の公開フォルダを確認する。

```
user01@ubuntu:~$ showmount -e 192.168.137.55
Export list for 192.168.137.55:
/usr/local 192.168.137.0/24
/srv/nfs   192.168.137.0/24
user01@ubuntu:~$
```

PCの/mntフォルダに/usr/localをマウントする。

```
user01@ubuntu:~$ sudo mount 192.168.137.55:/usr/local /mnt
```

これにより一連のインストール先を　/mnt　とする。

```
user01@ubuntu:~$ ls /mnt
bin  etc  games  include  lib  man  sbin  share  src
user01@ubuntu:~$
```

# 「付録3」

まず、x264をインストールする。

インストール先は　/mnt　とする。

x264フォルダに移動し　configure設定する。またffmpegはこのsharedライブラリを使用するので　--enable-shared を加える。

```
user01@ubuntu:/public/workffmpeg$ cd x264
user01@ubuntu:/public/workffmpeg/x264$ ./configure
 --host=arm-linux-gnue --enable-static --enable-shared
 --cross-prefix=/tools/arm-bcm2708/gcc-linaro-arm-linux-gnu
eabihf-raspbian-x64/bin/arm-linux-gnueabihf-
 --prefix=/mnt
```

make とインストールをする

```
user01@ubuntu:/public/workffmpeg/x264$ make
```

```
user01@ubuntu:/public/workffmpeg/x264$ sudo make install
```

# 「付録3」

次にalsa-libをインストールする。インストール先はターゲットつまりRaspiの/usr/localとする。(nfs接続でマウントされている　/mnt)

/alsa-libフォルダに移動し、configure 設定する。

```
user01@ubuntu:/public/workffmpeg$ cd alsa-lib-1.1.9/
user01@ubuntu:/public/workffmpeg/alsa-lib-1.1.9$ ./configure
 --host=arm-linux-gnueabihf --prefix=/mnt
```

make とインストールをする。

```
user01@ubuntu:/public/workffmpeg/alsa-lib-1.1.9$ make
```

```
user01@ubuntu:/public/workffmpeg/alsa-lib-1.1.9$ sudo make install
```

# 「付録3」

/mnt には一連のファイルが仕上がる。

```
user01@ubuntu:/mnt$ ls
bin  etc  games  include  lib  man  sbin  share  src
user01@ubuntu:/mnt$ ls bin
aserver  x264
user01@ubuntu:/mnt$ ls lib
libasound.la     libasound.so.2.0.0  libx264.so.147  python2.7
libasound.so     libx264.a           pkgconfig       python3.5
libasound.so.2   libx264.so          pypy2.7
user01@ubuntu:/mnt$
```

./configure －－helpでオプションが確認できる。

```
user01@ubuntu:/public/workffmpeg/ffmpeg$ ./configure --help
```

セルフコンパイラでは設定できた以下の

--enable-omx-rpi --enable-omx はコンフィグできないため

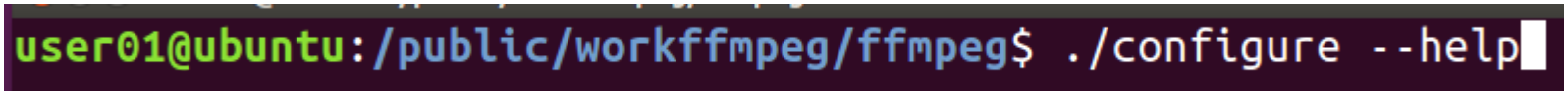
ffmpeg起動で h264_omxは使用できない。(mjpegは可能)

またffplayは生成されない。

# 「付録3」

/public/workffmpeg/ffmpeg　フォルダに移動して
./configure　－－helpでオプションが確認できる。

セルフコンパイラでは設定できた以下の
--enable-omx-rpi　--enable-omx はコンフィグできないため
ffmpeg起動で　h264_omxは使用できない。(mjpegは可能)
またffplayは生成されない。



# 「付録3」

/public/workffmpeg/ffmpegに移動して configure設定する。

```
user01@ubuntu:/public/workffmpeg/ffmpeg$ ./configure --prefix=/mnt
 --cross-prefix=/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-r
aspbian-x64/bin/arm-linux-gnueabihf- --enable-cross-compile
 --arch=armel --target-os=linux --enable-gpl --enable-libx264
 --enable-nonfree --extra-cflags="-I/mnt/include"
 --extra-ldflags="-L/mnt/lib" --extra-libs=-ldl
```

makeする。

```
License: nonfree and unredistributable
Creating configuration files ...

WARNING: using libx264 without pkg-config
user01@ubuntu:/public/workffmpeg/ffmpeg$ make
```

PCの性能にもよるが、約10分でコンパイルが終了する。

インストールする。

```
user01@ubuntu:/public/workffmpeg/ffmpeg$ sudo make install
```

# 「付録3」

Raspiの/usr/localのインストール状況を確認する。

```
pi@raspberrypi:~ $ cd /usr/local/
pi@raspberrypi:/usr/local $ ls
bin  etc  games  include  lib  man  sbin  share  src
pi@raspberrypi:/usr/local $ ls -al ./bin
合計 32540
drwxr-xr-x  2 root staff     4096  8月  2 15:58 .
drwxrwsr-x 10 root staff     4096  4月  8 18:37 ..
-rwxr-xr-x  1 root root     85309  8月  2 14:56 aserver
-rwxr-xr-x  1 root root  16114656  8月  2 15:58 ffmpeg
-rwxr-xr-x  1 root root  16027712  8月  2 15:58 ffprobe
-rwxr-xr-x  1 root root   1074317  8月  2 14:47 x264
pi@raspberrypi:/usr/local $
```

# 「付録3」

Raspiの/usr/localのインストール状況を確認する。（つづき）

```
pi@raspberrypi:/usr/local $ ls -al ./lib
合計 156228
drwxr-xr-x  6 root staff     4096  8月  2 16:00 .
drwxrwsr-x 10 root staff     4096  4月  8 18:37 ..
-rwxr-xr-x  1 root root       937  8月  2 14:56 libasound.la
lrwxrwxrwx  1 root root        18  8月  2 14:56 libasound.so -> libasound.so.2.0.0
lrwxrwxrwx  1 root root        18  8月  2 14:56 libasound.so.2 -> libasound.so.2.0.0
-rwxr-xr-x  1 root root   3701551  8月  2 14:56 libasound.so.2.0.0
-rw-r--r--  1 root root  90948676  8月  2 15:59 libavcodec.a
-rw-r--r--  1 root root   1197288  8月  2 15:58 libavdevice.a
-rw-r--r--  1 root root  19163226  8月  2 15:58 libavfilter.a
-rw-r--r--  1 root root  37175964  8月  2 15:59 libavformat.a
-rw-r--r--  1 root root   2287500  8月  2 16:00 libavutil.a
-rw-r--r--  1 root root    170368  8月  2 15:59 libpostproc.a
-rw-r--r--  1 root root    390428  8月  2 15:59 libswresample.a
-rw-r--r--  1 root root   2836384  8月  2 16:00 libswscale.a
-rw-r--r--  1 root root   1075376  8月  2 14:48 libx264.a
lrwxrwxrwx  1 root root        14  8月  2 14:48 libx264.so -> libx264.so.147
-rwxr-xr-x  1 root root    978425  8月  2 14:48 libx264.so.147
drwxr-xr-x  2 root root      4096  8月  2 16:00 pkgconfig
drwxrwsr-x  3 root staff     4096  4月  8 19:11 pypy2.7
drwxrwsr-x  4 root staff     4096  4月  8 18:56 python2.7
drwxrwsr-x  3 root staff     4096  4月  8 18:50 python3.5
pi@raspberrypi:/usr/local $
```

# 「付録3」

試しにffmpeg でwebカメラ映像、音声を記録する。

webカメラを確認する。

```
pi@raspberrypi:~ $ ls /dev/video0
/dev/video0
```

音声入力デバイスの確認

$ arecord -l

```
pi@raspberrypi:~ $ arecord -l
**** ハードウェアデバイス CAPTURE のリスト ****
カード 1: U0x46d0x8ca [USB Device 0x46d:0x8ca],
デバイス 0: USB Audio [USB Audio]
  サブデバイス: 1/1
  サブデバイス #0: subdevice #0
pi@raspberrypi:~ $
```

音声のデバイス:カード番号　1、 デバイス番号　0

設定名は　hw:1,0　となる。

# 「付録3」

映像入力デバイスの確認

$ v4l2-ctl --list-device

```
pi@raspberrypi:~ $ v4l2-ctl --list-device
bcm2835-codec (platform:bcm2835-codec):
        /dev/video10

UVC Camera (046d:08ca) (usb-3f980000.usb-1.3):
        /dev/video0
        /dev/video1
```

映像デバイスは　/dev/video0

# 「付録3」

out.mkv のファイル名で録画する。

$ ffmpeg −f v4l2 −i /dev/video0 −f alsa −ac 1 −i hw:1,0 −c:v mjpeg −c:a aac −f matroska out.mkv

```
pi@raspberrypi:~ $ ffmpeg -f v4l2 -i /dev/video0 -f alsa -ac 1
 -i hw:1,0 -c:v mjpeg -c:a aac -f matroska out.mkv
```

以下のエラーが出る場合は

libx264.so.147 を認識させる。

```
ffmpeg: error while loading shared libraries: libx264.so.147: ca
nnot open shared object file: No such file or directory
pi@raspberrypi:~ $
```

# 「付録3」

libx264.so.152 は/usr/local/lib にあり、この記述はlibc.confにある。

```
pi@raspberrypi:~ $ ls /usr/local/lib
libasound.la         libavcodec.a   libavutil.a      libx264.a        pypy2.7
libasound.so         libavdevice.a  libpostproc.a    libx264.so       python2.7
libasound.so.2       libavfilter.a  libswresample.a  libx264.so.147   python3.5
libasound.so.2.0.0   libavformat.a  libswscale.a     pkgconfig
pi@raspberrypi:~ $
```

/etc/ld.so.conf.d/libc.confを編集する。

```
pi@raspberrypi:~ $ cd /etc/ld.so.conf.d
pi@raspberrypi:/etc/ld.so.conf.d $ ls
00-vmcs.conf                fakeroot-arm-linux-gnueabihf.conf
arm-linux-gnueabihf.conf    libc.conf
pi@raspberrypi:/etc/ld.so.conf.d $ nano libc.conf
```

```
# libc default configuration
/usr/local/lib
```

# 「付録3」

$ sudo　ldconfig を実行する。

```
pi@raspberrypi:~ $ sudo ldconfig
```

# 「付録3」

再度実行

```
pi@raspberrypi:~ $ ffmpeg -f v4l2 -i /dev/video0 -f alsa -ac 1
 -i hw:1,0 -c:v mjpeg -c:a aac -f matroska out.mkv
ffmpeg version 4.2.git Copyright (c) 2000-2019 the FFmpeg devel
opers
  built with gcc 4.8.3 (crosstool-NG linaro-1.13.1+bzr2650 - Li
naro GCC 2014.03) 20140303 (prerelease)
```

^Cにて終了させる。

```
frame=   39 fps= 39 q=8.1 size=     188kB time=00:00:07.
frame=   46 fps= 31 q=8.6 size=     216kB time=00:00:08.
frame=   54 fps= 27 q=9.3 size=     243kB time=00:00:08.
frame=   61 fps= 24 q=9.7 size=     262kB time=00:00:09.
frame=   69 fps= 23 q=10.2 size=     287kB time=00:00:09
frame=   76 fps= 22 q=10.7 size=     313kB time=00:00:10
frame=   84 fps= 21 q=11.3 size=     338kB time=00:00:10
```

# 「付録3」

記録したout.mkv はVLCなどで確認できる。

ただし、raspberryPiでは再生できない。（性能不足？）

PCなどにコピーして確認可能。

音声をイヤホンジャックで出力する場合は

http://www.yam-web.net/raspberry-pi/music.html

を参考に

$ amixer  cset  numid=3 1

```
pi@raspberrypi:~ $ amixer cset numid=3 1
```